# 38장 브라우저의 렌더링 과정

- **파싱**
  - 파싱(구문 분석)은 프로그래밍 언어의 문법에 맞게 작성된 텍스트 문서를 읽어 들여 실행하기 위해 텍스트 문서의 문자열을 토큰으로 분해(어휘 분석)하고,
  - 토큰에 문법적 의미와 구조를 반영하여 트리 구조의 자료구조인 파스 트리를 생성하는 일련의 과정을 말한다.
  - 일반적으로 파싱이 완료된 이후에는 파스 트리를 기반으로 중간언어인 바이트코드를 생성하고 실행한다.
- **렌더링**
  - 렌더링은 HTML, CSS, 자바스크립트로 작성된 문서를 파싱하여 브라우저에 시각적으로 출력하는 것을 말한다.

## 브라우저의 렌더링 과정

**1.** 브라우저는 HTML, CSS, 자바스크립트, 이미지, 폰트 파일 등 렌더링에 필요한 리소스를 요청하고 서버로부터 응답을 받는다.

**2.** 브라우저의 렌더링 엔진은 서버로부터 응답된 HTML과 CSS를 파싱하여 DOM과 CSSOM을 생성하고 이들을 결합하여 렌더 트리를 생성한다.

**3.** 브라우저의 자바스크립트 엔진은 서버로부터 응답된 자바스크립트를 파싱하여 AST를 생성하고 바이트코드로 변환하여 실행한다.

이때 자바스크립트는 DOM API를 통해 DOM이나 CSSOM을 변경할 수 있다.

변경된 DOM과 CSSOM은 다시 렌더 트리로 결합된다.

**4.** 렌더 트리를 기반으로 HTML 요소의 레이아웃(위치와 크기)을 계산하고 브라우저 화면에 HTML 요소를 페인팅한다.

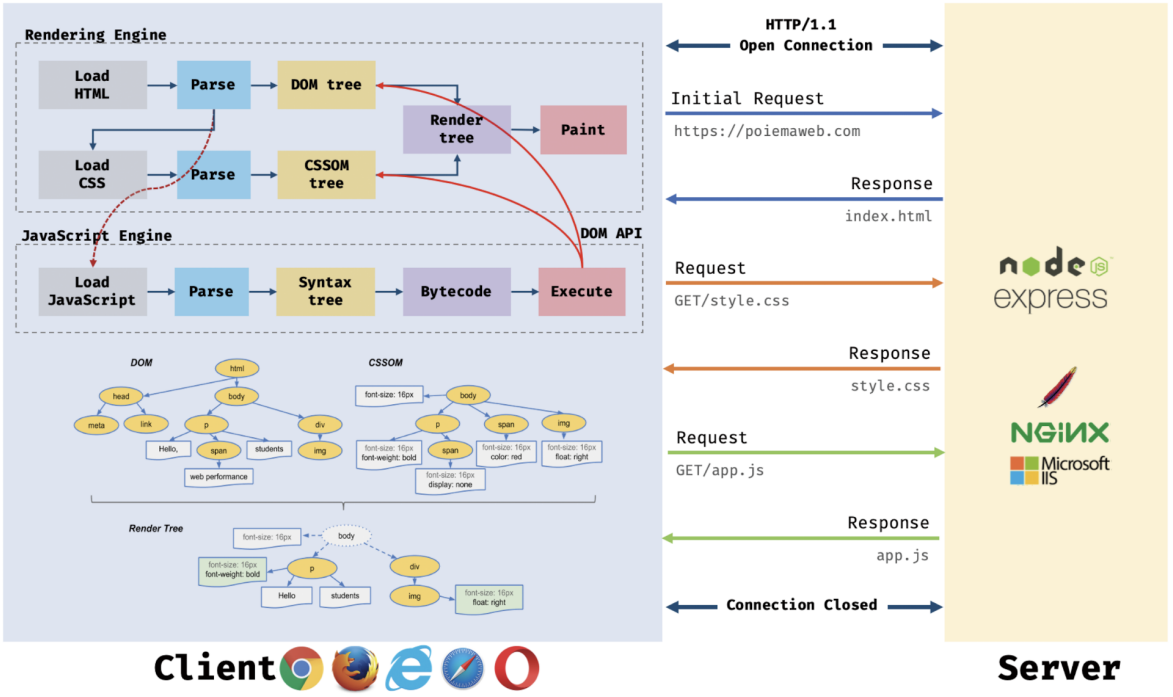


그림 38-1 브라우저의 렌더링 과정

# 38.1 요청과 응답

- **브라우저의 핵심 기능은 필요한 리소스를 서버에 요청하고 서버로부터 응답받아 브라우저에 시각적으로 렌더링 하는 것**이다.
- 즉, 렌더링에 필요한 리소스는 모두 서버에 존재하므로 필요한 리소스를 서버에 요청하고 서버가 응답한 리소스를 파싱하여 렌더링하는 것이다.
- 서버에 요청을 전송하기위해 브라우저는 주소창을 제공한다.**URL의 호스트 이름리 DNS를 통해 IP 주소로 변환되고이 IP주소를 갖는 서버에게 요청을 전송**한다.

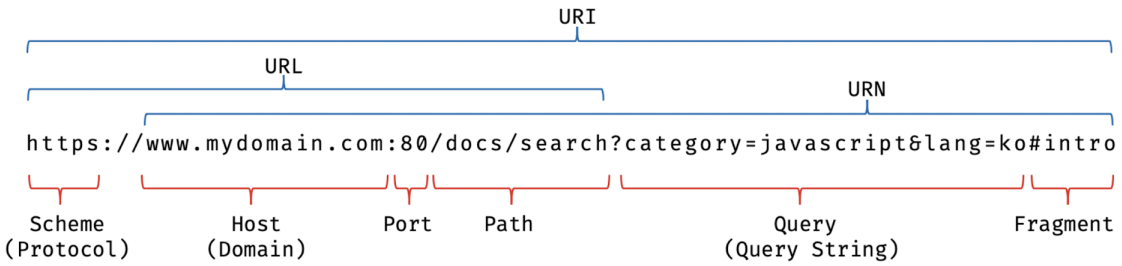


그림 38-2 URI[Uniform Resource Identifier][6]

- 일반적으로 서버는 루트 요청에 대해 암묵적으로 index.html을 응답하도록 기본 설정 되어있다.

즉 `https://poiemaweb.com`은 `https://poiemaweb.com/index.html`과 같은 요청이다.

- 다른 정적 파일을 서버에 요청하려면 정적 파일의 경로(서버의 루트 폴더 기준)와 파일 이름을 URL 호스트 뒤에 패스에 기술하여 서버에 요청한다.
- 서버에게 동적으로 서버에 정적/동적 데이터를 요청할 수도 있다.

---

# 38.2 HTTP 1.1과 HTTP 2.0

- **HTTP는 브라우저와 서버가 통신하기 위한 프로토콜(규약)**

## HTTP/1.1

- 기본적으로 커넥션당 하나의 요청과 응답만 처리한다.
- 즉 리소스 요청이 개별적으로 전송되고 응답 또한 개별적으로 전속된다.
- 요청할 리소스의 개수에 비례하여 응답 시간도 증가하는 단점이 있다.

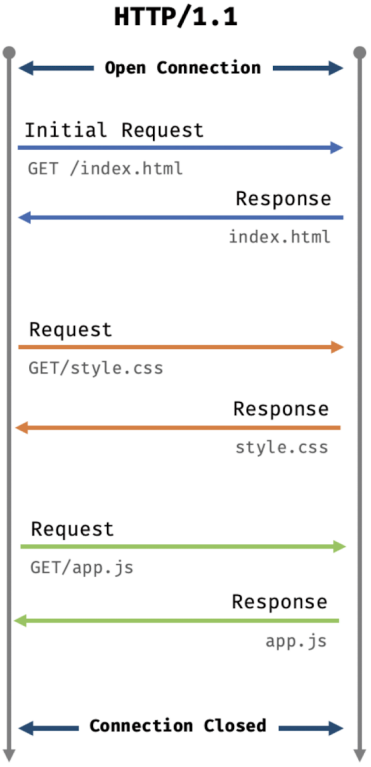


그림 38-4 HTTP/1.1

## HTTP/2.0

- 커넥션당 여러 개의 요청과 응답, 즉 **다중 요청/응답이 가능**하다.

- 따라서 여러 리소스의 동시 전송이 가능하므로 **HTTP/1.1에 비래 페이지 로드 속도가 약 50% 빠르다**고 알려져 있다.

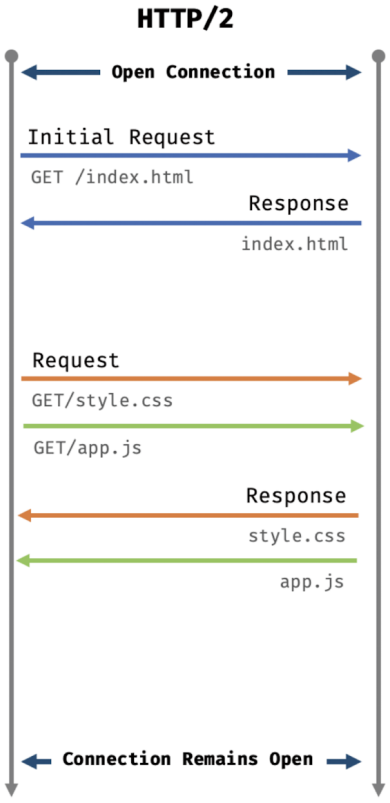


그림 38–5 HTTP/2

# 38.3 HTML 파싱과 DOM 생성

- 브라우저의 요청에 서버가 응답한 HTML 문서는 문자열로 이루어진 순수한 텍스트
- HTML 문서를 브라우저가 이해할 수 있는 자료구조(객체)로 변환하여 메모리에 저장해야 렌더링이 가능하다.

```
<!DOCTYPE html>
<html lang="kr">
  <head>
    <meta charset="UTF-8" />
    <link rel="stylesheet" href="style.css" />
    <title>Title</title>
  </head>
  <body>
    <ul>
```

```
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script src="app.js"></script>
  </body>
</html>
```

- 브라우저의 렌더링 엔진은 아래와 같은 과정을 통해 응답받은 HTML 문서를 파싱하여 브라우저가 이해할 수 있는 자료구조인 DOM을 생성한다.

1. **서버에 존재하던 HTML 파일이 브라우저의 요청에 의해 응답된다.**

   이때 서버는 브라우저가 요청한 HTML 파일을 읽어 들여 메모리에 저장한 다음 메모리에 저장된 바이트(2진수)를 인터넷을 경유하여 응답한다.

2. **브라우저는 서버가 응답한 HTML 문서를 바이트(2진수) 형태로 응답받는다.**

   그리고 응답된 바이트 형태의 HTML 문서는 meta 태그의 charset 어트리뷰트에 의해 지정된 인코딩 방식(UTF-8)을 기준으로 문자열로 변환된다.

3. 문자열로 변환된 HTML문서를 읽어들여 문법적 의미를 갖는 코드의 최소단위인 **토큰들로 분해**한다.

4. 각 토큰들을 **객체로 변환하여 노드들을 생성**한다.

   토큰의 내용에 따라 문서노드, 요소노드, 어트리뷰트 노드, 텍스트노드가 생성된다. 노드는 이후에 DOM을 구성하는 기본 요소가 된다.

5. **모든 노드들을 트리 자료구조로 구성**한다.

   이 노드들로 구성된 트리 자료구조를 DOM 이라고 부른다.

- 즉, **DOM은 HTML 문서를 파싱한 결과물**

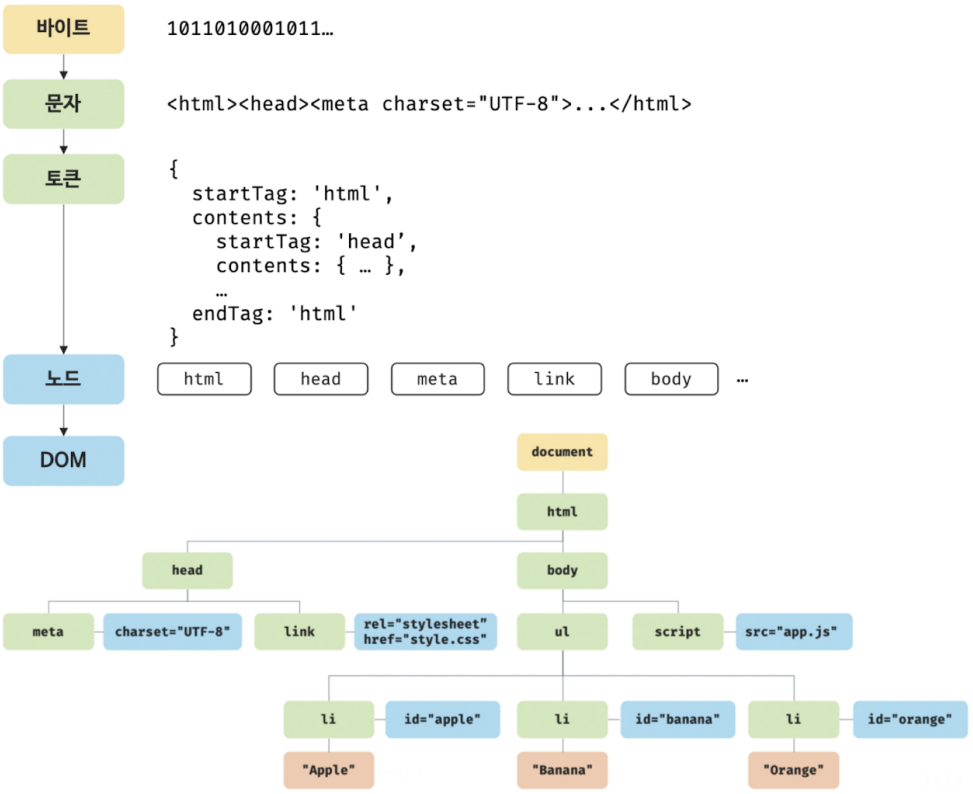


그림 38–6 HTML 파싱과 DOM 생성

# 38.4 CSS 파싱과 CSSOM 생성

- 렌더링 엔진은 DOM을 생성해 나가다가 **CSS 로드를 하는 link 태그나 style태그를 만나면 DOM 생성을 일시 중단**한다.
- 그리고 link 태그의 href 어트리뷰트에 지정된 CSS 파일을 서버에 요청하여

  로드한 CSS파일이나 style 태그 내의 CSS 파싱과정(바이트->문자->토큰->노드->CSSOM)을 거치며 해석하여 CSSOM을 생성한다.
- 이후 CSS파싱이 완료되면 HTML 파싱이 중단된 지점부터 다시 HTML 파싱하기 시작하여 DOM 생성을 재개한다.

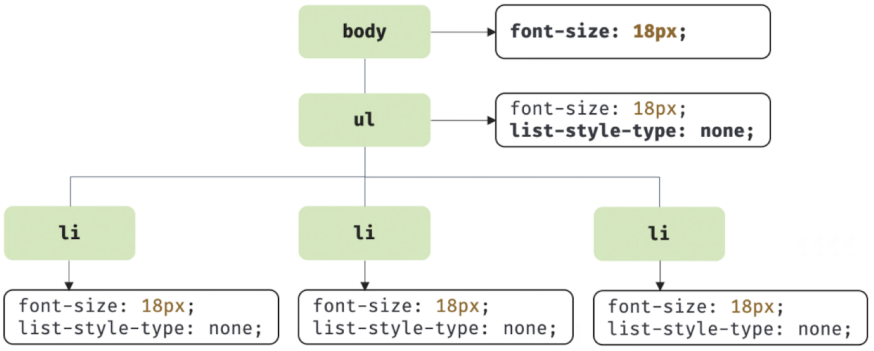


그림 38-7 CSSOM 생성

# 38.5 렌더 트리 생성

- **DOM과 CSSOM은 렌더링을 위해 렌더 트리로 결합**된다.
- 렌더 트리는 렌더링을 위한 트리 구조의 자료구조다.
  - 따라서 브라우저 화면에 렌더링되지 않는 노드(예:meta, script 태그 등)와 CSS에 의해 비표시(예: display:none)되는 노드들은 포함하지 않는다.
- **렌더 트리는 브라우저 화면에 렌더링되는 노드만으로 구성**된다.

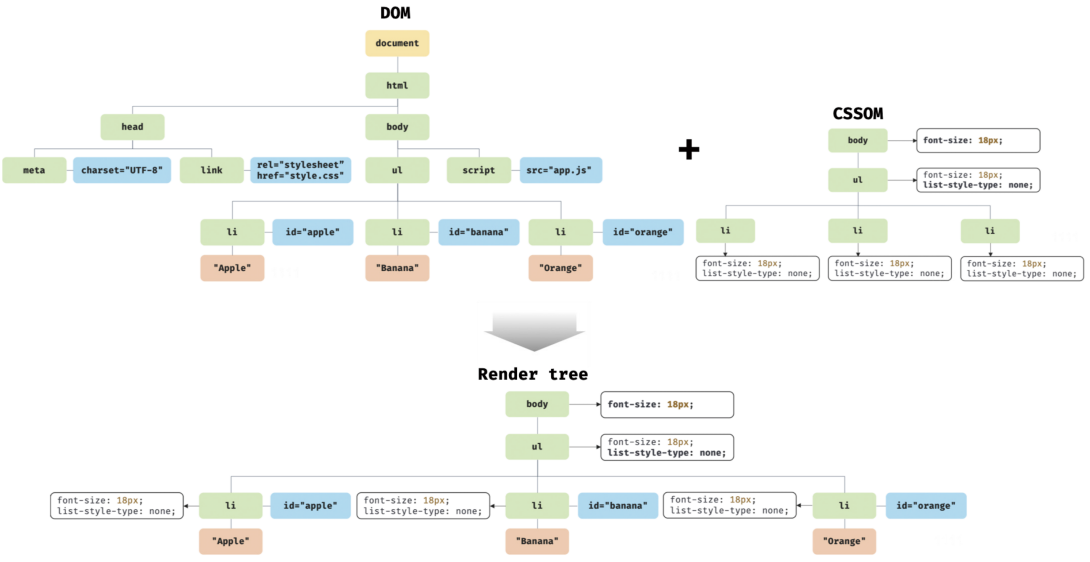


그림 38-8 렌더 트리 생성

- 이후 완성된 렌더트리는 각 HTML 요소의 레이아웃(위치와 크기)을 계산하는 데 사용되며 브라우저 화면에 픽셀을 렌더링하는 페인팅 처리에 입력


그림 38-9 렌더 트리와 레이아웃/페인트

- 브라우저의 렌더링과정은 다음과 같은 경우 반복해서 실행된다.
  - 자바스크립트에 의한 노드 추가 또는 삭제
  - 브라우저 창의 리사이징에 의한 뷰포트 크기 변경
  - HTML 요소의 레이아웃(위치,크기)에 변경을 발생시키는 width,heigth,margin,padding 등의 스타일 변경
- 레이아웃 계산과 페인팅을 다시 실행하는 **리렌더링은 비용이 많이 드는, 즉 성능이 악영향을 주는 작업이다.**
- **따라서 가급적 리렌터링이 빈번하게 발생하지 않도록 주의할 필요가 있다.**

---

# 38.6 자바스크립트 파싱과 실행

- **자바스크립트 코드에서 DOM API를 사용하면 이미 생성된 DOM을 동적으로 조작**할 수 있다.
- 자바스크립트 코드를 파싱하기 위해 자바스크립트 엔진에 제어권을 넘긴다.

  이후 자바스크립트 파싱과 실행이 종료되면 렌더링 엔진으로 다시 제어권을 넘겨 HTML 파싱이 중단된 지점부터 다시 HTML 파싱을 시작하여 DOM 생성을 재개한다.
- 자바스크립트 파싱과 실행은 **브라우저의 렌더링 엔진이 아닌 자바스크립트 엔진이 처리**한다.
- 자바스크립트 엔진은 자바스크립트 코드를 파싱하여 CPU가 이해할 수 있는 저수준 언어로 변환하고 실행하는 역할을 한다.
- 렌더링 엔진으로부터 제어권을 넘겨받은 자바스크립트 엔진은 자바스크립트 코드를 파싱하기 시작한다.

  HTML과 CSS를 파싱하여 DOM과 CSSOM을 생성하듯이 **자바스크립트 엔진은 자바스크립트를 해석하는 AST(추상적 구문 트리)를 생성**한다.

  그리고 AST를 기반으로 인터프리터가 실행할 수 있는 중간 코드인 바이트 코드를 생성하여 실행한다.

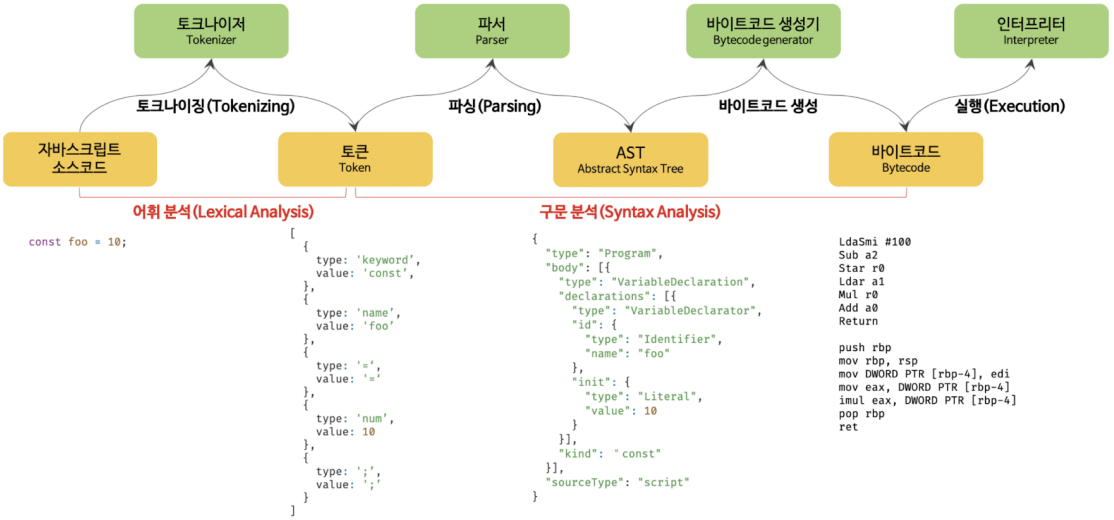


그림 38-10 자바스크립트 파싱과 실행

### 토크나이징

- 단순한 문자열인 자바스크립트 소스코드를 어휘 분석하여 문법적 의미를 갖는 코드의 최소 단위인 토큰들로 분해

### 파싱

- 토큰들의 집합을 구분 분석하여 AST(추상적 구문 트리)를 생성

### 바이트코드 생성과 실행

- 파싱의 결과물로서 생성된 AST는 인터프리터가 실행할 수 있는 중간 코드인 바이트코드로 변환되고 인터프리터에의해 실행

---

# 38.7 리플로우와 리페인트

- 만약 자바스크립트 코드에 DOM이나 CSSOM을 변경하는 DOM API가 사용되는 경우 DOM이나 CSSOM이 변경된다.
- 이때 **변경된 DOM과 CSSOM은 다시 렌더 트리로 결합되고 변경된 렌더 트리를 기반으로 레이아웃과 페인트 과정을 거쳐 브라우저의 화면에 다시 렌더링**한다.

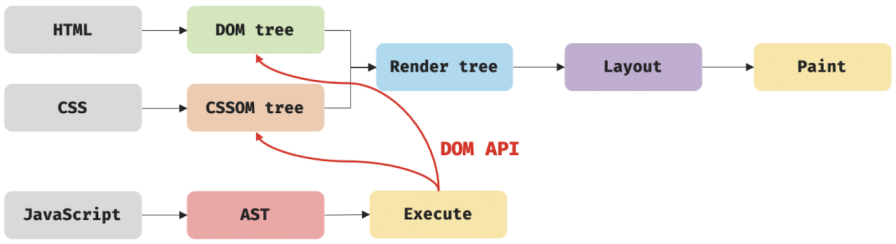


그림 38-12 DOM API에 의한 리플로우, 리페인트

- **리플로우는**

  레이아웃 계산을 다시 하는 것을 말하며, 노드 추가/삭제, 요소의 크기/ 위치 변경, 윈도우 리사이징 등 레이아웃에 영향을 주는 변경이 발생한 경우 실행

- **리페인트는**

  재결합된 렌더 트리를 기반으로 다시 페인트를 하는 것

- 레이아웃에 영향이 없는 변경은 리플로우 없이 리페인트만 실행

---

# 38.8 자바스크립트 파싱에 의한 HTML 중단

- 자바스크립트 코드에서 **DOM이나 CSSOM을 변경하는 DOM API를 사용할 경우 DOM이나 CSSOM이 이미 생성되어 있어야한다.**
- 생성이 완료되지 않은 상태라면 문제가 발생할 수 있다.
- 이러한 문제를 회피하기 위해 **body 요소의 가장 아래에 자바스크립트를 위치시키는 것**은 좋은 아이디어다.
  - DOM이 완성하지 않은 상태에서 자바스크립트가 DOM을 조작하면 에러가 발생할 수 있다.
  - 자바스크립트 로딩/파싱/실행으로 인해 HTML 요소들의 렌더링에 지장받는 일이 발생하지 않아 페이지 로딩 시간이 단축된다.

---

# 38.9 script 태그의 async/defer 어트리뷰트

- 자바스크립트 파싱에 의한 DOM 생성이 중단되는 문제를 근본적으로 해결하기 위해 HTML5부터 추가되었다.
- async와 defer 어트리뷰트는 다음과 같이 src 어트리뷰트를 통해 외부 자바스크립트 파일을 로드하는 경우에만 사용할 수 있다.

  즉, src 어트리뷰트가 없는 인라인 자바스크립트에는 사용할 수 없다.

```
<script async src="extern.js"></script>
<script defer src="extern.js"></script>
```

- **async와 defer 어트리뷰트를 사용하면 HTML 파싱과 외부 자바스크립트 파일의 로드가 비동기적으로 동시에 진행**된다.

## async 어트리뷰트

- HTML 파싱과 외부 자바스크립트 파일의 로드가 비동기 적으로 진행된다.
- 단, **자바스크립트의 파싱과 실행은 자바스크립트 파일의 로드가 완료된 직후 진행되며, 이때 HTML 파싱이 중단**


그림 38-14 script 태그의 async 어트리뷰트

- 여러개의 script 태그에 async 어트리뷰트를 지정하면 script 태그의 순서와는 상관없이 로드가 완료된 자바스크립트부터 먼저 실행되므로 순서가 보장되지 않는다.
- 따라서 **순서 보장이 필요한 script 태그에는 async 어트리뷰트를 지정하지 않아야한다.**

## defer 어트리뷰트

- HTML 파싱과 외부 자바스크립트 파일의 로드가 비동기 적으로 진행된다.
- 단, 자바스크립트의 파싱과 실행은 HTML 파싱이 완료된 직후, 즉 DOM 생성이 완료된 직후(이때 DOMContentLoaded 이벤트가 발생한다.)

> DOMContentLoaded 이벤트
>
> DOM 트리가 다 만들어진 후에 돔에 접근이 가능하기때문에,
>
> 돔이 생성되기전 돔을 조작하는 자바스크립트 코드가 실행되어 원하지 않는 결과를 내는것을 막을 수 있다.

- 따라서 DOM 생성이 완료된 이후 실행되어야 할 자바스크립트에 유용하다.


그림 38-15 script 태그의 defer 어트리뷰트